Date of Deposit: July 24, 2001

APPLICATION FOR LETTERS PATENT
OF THE UNITED STATES

NAME OF INVENTORS: REGINA SCHMITT
Herbstäckerweg 5
91056 Erlangen
Germany

PETER WAGNER
Kellerstraße 7
91217 Hersbruck
Germany

TITLE OF INVENTION: FLOWCHART PROGRAMMING FOR INDUSTRIAL CONTROLLERS, IN PARTICULAR MOTION CONTROLLERS

TO WHOM IT MAY CONCERN, THE FOLLOWING IS
A SPECIFICATION OF THE AFORESAID INVENTION

2000 P 14826 US
Express Mail Label No EL 728053197US

2000 P 14826 US

FLOWCHART PROGRAMMING FOR INDUSTRIAL CONTROLLERS, IN PARTICULAR MOTION CONTROLLERS

[0001] This invention relates to a method for programming industrial controllers, in particular motion controllers, where the user links graphical elements, in particular control structures and function blocks, with a graphical editor to form a flowchart that can be visualized on a display device.

[0002] Furthermore, this invention relates to a device for programming industrial controllers, in particular motion controllers, where control structures and function blocks can be linked by a user with a graphical editor to form a flowchart that can be visualized on a display device.

[0003] In the industrial environment, it is known that graphical input aids and a display screen can be used for visualizing the control of the motion of a processing machine or a production machine (Hans D. Kief: NC/CNC Handbuch 2000 [NC/CNC Handbook 2000], Hansa Verlag, p. 254, fig. 7 and p. 327, fig. 6). The basic elements of graphical structure diagrams and flowcharts are defined in German Industrial Standard DIN 66,001.

[0004] In addition, it is stated in the article "Visual Languages - an Inexorable Trend in Industry" (Josef Hübl, PLCs/IPCs/Drives - Convention Volume, pp. 88-95, November 23-25, 1999, Nuremberg, Verlag, Hüthig GmbH, Heidelberg) that control flowcharts and data-flow diagrams for control of automation functions may be created with the help of graphical editors.

[0005] The graphical input means and graphical editors available today for programming industrial controllers support only dedicated programming of either

the controller of an industrial process, i.e., a programmable controller (PLC) function, or the motion controller of a processing machine or production machine. Creation of programs for both fields of applications is not adequately supported by the existing flowchart editors.

[0006] Another disadvantage of the flowchart editors used today for programming industrial automation functions is that the diagrams created with these editors are either converted directly to executable processor code or to ASCII code, and must subsequently be interpreted in the respective target system through a run time-intensive process. This system is not just inflexible with regard to porting and transfer of programs to other systems or machines, but does also restrict the user's debugging options.

[0007] Additional disadvantages of existing flowchart editors is that only a limited, rigid and inflexible library of icons is available, and that the processing sequence of icons and the corresponding function blocks is predetermined. Further, existing flowchart editors frequently offer only a limited number of possibilities for formulating synchronization mechanisms, although such mechanisms are frequently required, in particular for programming applications in industrial automation.

[0008] The purpose of this invention is to create a method whereby support is provided for the user for programming the controller of technical processes (programmable controller functionality) and also for programming motion controllers (motion functionality). The user should be offered the greatest possible flexibility in programming and in particular the user should be supported by adaptive mechanisms.

[0009] According to this invention, the object defined above is achieved for a method of the type defined in the preamble by carrying out the process steps in the following succession:

a) generating a textual language from the flowchart,

b) converting the textual language in a processor-independent pseudo-code,

c) loading the processor-independent pseudo-code into the controller,

d) converting the processor-independent pseudo-code into executable processor code, where adequate programming language commands are made available to the user in the flowchart editor as a function of the underlying machine configuration or hardware configuration.

[0010] That a textual language is generated from the flowchart diagrams in an intermediate step, provides the user the option of performing plausibility checks already at this level of the textual language. However, the user can also tie other language elements, already present in the textual language to the application. Due to the fact that the textual language is converted in a processor-independent pseudo-code in another intermediate step, the aforementioned flexibility for the user is largely retained. The user can also perform plausibility checks or debugging at this pseudo-code level. The processor code that ultimately runs in the controller is generated from the processor-independent pseudo-code, so the target of the application is not specified until a late point in time. Furthermore, different target hardware units can easily be operated due to the intermediate steps in the generation of the code.

[0011] Another advantage of this invention is that current and adequate programming language commands are made available for the user in the form of icons in the flowchart editor for each basic machine design and/or hardware configuration. This provides a programming environment, that is adapted to the basic hardware and, thus, meets the existing requirements and boundary conditions in an optimal manner. The library of icons of the flowchart editor will

be automatically adapted to the given hardware specifications, e.g., the basic machine configuration.

[0012] In the first advantageous embodiment of the present invention, graphical elements containing the function interface of corresponding subprograms may be generated in a flowchart representation. The elements are generated automatically in the flowchart view from user-defined subprograms. Hence, it is possible to generate icons and the respective masks automatically from the system, from existing subprograms of the textual language, or from additional optimal sub-programs generated by the machine manufacturer and made available to the user through the flowchart editor. The function interface and the transfer parameters of the subprograms of the textual language are generated automatically for the flowchart icons. Due to these mechanisms, original equipment manufacturers may transfer subprograms that are already prepared in textual language into the flowchart editor. Thus, an adapted and expanded library of icons is made available to the end user for flowchart programming.

[0013] In another embodiment of the present invention, the automatically generated graphical elements may be used as language elements of the flowchart. This increases the library of icons of flowchart elements available to the user, i.e., icons, due to the fact that these automatically generated icons can be used as normal language elements of the flowchart editor. This increases the user's flexibility and ease of expression with regard to programming of applications.

[0014] In another advantageous embodiment of this invention, the textual language is "structured text" according to the international standard IEC 6-1131 (International Electrotechnical Commission, 1992, Programmable controllers - Part 1: General information, IEC 61131-1, standard published by International Electrotechnical Commission). Due to the fact that IEC 6-1131 is a standardized language at the level of the textual language, it is readily possible to exchange it

or combine it with other programming systems. In addition, when IEC 6-1131 is used as an intermediate language, it becomes easier to port to different target systems.

[0015] In another advantageous embodiment of this invention a user can transition between textual language, the contact plan and/or the function plan as forms of representation for expressing conditions as desired, since IEC 6-1131 is used as textual language on the structured text level. This provides increased flexibility for the user, and is an advantage in particular for formulating conditions, because the user can select the form of representation or description in which he or she is most experienced or which is most appropriate to the underlying problem. A user will generally use contact plans and/or function plans for representing binary links and structured text for formulating arithmetic calculations.

[0016] In another advantageous embodiment of the present invention, at least one loop and/or at least one parallel branch may be used as a language element in the flowchart view. In the flowchart editors conventionally used today, loops and frequently also branches are represented with the help of jump marks. However, due to the use jumps and the respective targets marks, the program design may become complicated and difficult to reproduce (a "Go To problem"). Due to the fact that loops and parallel branches are available to the user as separate language elements, program generation and also the readability of programs are greatly simplified it becomes easier to develop and read the programs.

[0017] In another advantageous embodiment of the present invention, the individual commands are started in the same interpolator cycle within the respective parallel branch. Due to the fact that all the branches of the parallel branch language function are operated in the same interpolator cycle, it is possible to perform quasi-parallel processing of the commands contained in the

individual branches of a parallel branching construct. In addition to sequential processing, therefore, parallel processing of commands is also made possible and is supported by adequate programming language commands in the programming environment for the user.

[0018] In another advantageous embodiment of the present invention, the parameters for function blocks may be set by mask input in the flowchart view. Thus, the user may set parameters in a way that is simple and easy to understand. For each type of function block there are standard masks, which only allow a user to make the parameter inputs that are possible for the current type. The risk of defective inputs is reduced by this context sensitivity.

[0019] In another advantageous embodiment of the present invention, function blocks are combined into modules in the flowchart view. These modules also appear as function blocks. This increases the simplicity of program execution in the flowchart for the user. A user can combine function blocks that belong together logically into one module, and can encapsulate them there, in which case this module also appears as a function block in the flowchart editor, that is, as an icon. However, this mechanism of combining and encapsulation does not merely make it simpler to run the program, but also allows for the program to be structured in this way.

[0020] In another advantageous embodiment of the present invention, modules are interleaved in the flowchart view. This means that a module may contain one or more modules as an element. Modules may subsequently be used as "subprograms" in other modules, thus, increasing the simplicity and structure of the program execution in the flowchart.

[0021] In another advantageous embodiment of the present invention, the user may employ multiple instructions in the function blocks for the allocation of variables in the flowchart view. The user can enter multiple variable instructions

in succession into one function block, that is, into one icon, and does not need a new function block for each variable instruction. The user can also perform variable instructions, which logically belong to this function block, bundled in this one function block, which increases comprehensibility and supports the programming principle of high cohesion.

[0022] In another advantageous embodiment of the present invention, the function blocks representing the functions that require a certain amount of time, contain progression conditions in the flowchart view. Functions that claim a period of time include, for example, an approach to reference points, acceleration or axial positioning. Such functions and their interaction can be synchronized by the user with the help of the progression conditions. Thus, with the help of the progression conditions, a user has access to a synchronization mechanism that allows to synchronized complex motions and relationships among multiple axes.

[0023] In another advantageous embodiment of the present invention, the graphical elements of the flowchart are automatically positioned. When a user wants to represent a new icon in the flowchart editor, it is automatically positioned at the point that is the next, in order to correspond to the logical program sequence. This increases the user's efficiency, since one does not have to position the icons that one has generated.

[0024] In another advantageous embodiment of the present invention, the icons of the flowchart are automatically linked together. This also increases the operating efficiency of the user because need not to be linked the icons together manually.

[0025] In another advantageous embodiment of the present invention, the flowchart may be shown in a reduced or enlarged form in the display. Due to this zoom function, the diagrams are easier to comprehend for the user, and

furthermore, when the user is interested in certain program sequences, he can emphasize them graphically by enlarging them.

[0026] In another advantageous embodiment of the present invention, re-translation back into flowchart representation is possible because of markings in the textual language. Due to the use of syntactical and geometric information in the form of markings, it is possible to perform such reverse translation from the textual language into the flowchart view. The reverse translation option has the advantage for the user that changes entered at the level of the textual language can be implemented directly in the flowchart view through the flowchart editor and then become visible for the user in the flowchart diagrams. The user can then further process such retranslated programs with the help of the flowchart editor on the graphical level, and can subsequently generate control code from the said reconverted programs in the remaining procedure.

[0027] The objective defined above is achieved for a device by means of the following successive elements:

a) generating a textual language from the flowchart.

b) converting the textual language in a processor-independent pseudo-code.

c) loading the processor-independent pseudo-code into the controller.

d) converting the processor-independent pseudo-code into an executable processor code, where adequate programming language commands are made available to the user in the flowchart editor as a function of the underlying machine configuration or hardware configuration.

[0028] The user has the option of performing plausibility checks even at this level of textual language, because a textual language can be generated from the flowchart diagrams in an intermediate step. However, the user can also tie other language elements present in the textual language to his application. The

required flexibility for the user is preserved because of the fact that in another intermediate step, the textual language can be converted in a processor-independent pseudo-code. A user may also perform plausibility checks or debugging at this pseudo-code level. The processor code that ultimately runs in the controller can be generated from the processor-independent pseudo-code, and therefore the target of the application can be defined only at a late point in time. Furthermore, it is also easy to operate different types of target hardware due to the intermediate steps in code generation.

[0029] Another advantage of the present invention is that adequate programming language commands, are made available for the user in the form of icons in the flowchart editor, for each underlying machine configuration or machine design. A user then has a programming environment which is adapted to the underlying hardware and is thus optimal in meeting existing requirements and boundary conditions. The library of icons of the flowchart editor is thus automatically adapted to the given hardware specifications (e.g., the underlying machine configuration).

[0030] In another advantageous embodiment of this invention, graphical elements containing the function interface of the corresponding subprograms can be generated automatically in flowchart notation from the corresponding user-defined subprograms of the textual language. This enables automatic generation of icons and the respective masks of systems from existing subprograms of the textual language or from additional subprograms optionally introduced into the textual language by the machine manufacturer and made available to the user in the flowchart editor. The function interface and transfer parameters of the subprograms of the textual language are generated automatically for flowchart icons. Due to this mechanism, original equipment manufacturers can transfer subprograms in textual language to the flowchart

editor. An adapted and expanded library of icons is thus made available to the end user for flowchart programming.

[0031] In another advantageous embodiment of this invention, the automatically generated graphical elements may be used as language elements of the flowchart. This increases the library of flowchart elements available to the user, i.e., icons, due to the fact that these automatically generated icons can be used as normal language elements of the flowchart editor. It also increases the user's flexibility and ease of expression with regard to programming applications.

[0032] In another advantageous embodiment of this invention, the textual language is "structured text" according to the international standard IEC 6-1131 (International Electrotechnical Commission, 1992, Programmable controllers - Part 1: General information, IEC 61131-1, standard published by International Electrotechnical Commission). Due to the fact that IEC 6-1131 is a standardized language at the level of the textual language, it is readily possible to exchange it or combine it with other programming systems. In addition, when IEC 6-1131 is used as an intermediate language, it becomes easier to port to different target systems.

[0033] In another advantageous embodiment of the present invention, a user can transition between textual language, the contact plan and/or the function plan as forms of representation for expressing conditions as desired, since IEC 6-1131 is used as textual language on the structured text level. This provides increased flexibility for the user, and is an advantage in particular for formulating conditions, because the user can select the form of representation or description in which he or she is most experienced or which is most appropriate to the underlying problem. A user will generally use contact plans and/or function plans for representing binary links and structured text for formulating arithmetic calculations.

[0034] In another advantageous embodiment of the present invention, at least one loop and/or at least one parallel branch may be used as a language element in the flowchart view. In the flowchart editors conventionally used today, loops and frequently also branches are represented with the help of jump marks. However, due to the use of jumps and the respective targets marks, the program design may become complicated and difficult to reproduce (a "Go To problem"). Due to the fact that loops and parallel branches are available to the user as separate language elements, it becomes easier to develop and read the programs.

[0035] In another advantageous embodiment of the present invention, the individual commands are started in the same interpolator cycle within the respective parallel branch. Due to the fact that all the branches of the parallel branch language function language function are operated in the same interpolator cycle, it is possible to perform quasi-parallel processing of the commands contained in the individual branches of a parallel branching construct. In addition to sequential processing, therefore, parallel processing of commands is also made possible and is supported by adequate programming language commands in the programming environment for the user.

[0036] In another advantageous embodiment of the present invention, the parameters for function blocks may be set by mask input in the flowchart view. Thus, the user may set parameters in a way that is simple and easy to understand. For each type of function block there are standard masks, which only allow a user to make the parameter inputs that are possible for the current type. The risk of defective inputs is reduced by this context sensitivity.

[0037] In another advantageous embodiment of the present invention, function blocks are combined into modules in the flowchart view. These modules also appear as function blocks. This increases the simplicity of program execution in the flowchart for the user. A user can combine function blocks that belong

together logically into one module, and can encapsulate them there, in which case this module also appears as a function block in the flowchart editor, that is, as an icon. However, this mechanism of combining and encapsulation does not merely make it simpler to run the program, but also allows for the program to be structured in this way.

[0038] In another advantageous embodiment of the present invention, modules are interleaved in the flowchart view. This means that a module may contain one or more modules as an element. Modules may subsequently be used as "subprograms" in other modules, thereby, increasing the simplicity and structure of the program execution in the flowchart.

[0039] In another advantageous embodiment of the present invention, the user may employ multiple instructions in the function blocks for the allocation of variables in the flowchart view. The user can enter multiple variable instructions in succession into one function block, that is, into one icon, and does not need a new function block for each variable instruction. The user can also perform variable instructions, which logically belong to this function block, bundled in this one function block, which increases comprehensibility and supports the programming principle of high cohesion.

[0040] In another advantageous embodiment of the present invention, the function blocks representing the functions that require a certain amount of time, contain progression conditions in the flowchart view. Functions that claim a period of time include, for example, an approach to reference point, acceleration or axial positioning. Such functions and their interaction can be synchronized by the user with the help of the progression conditions. Thus, with the help of the progression conditions, a user has access to a synchronization mechanism that allows synchronized complex motions and relationships among multiple axes.

[0041] In another advantageous embodiment of the present invention, the graphical elements of the flowchart are automatically positioned. When a user wants to represent a new icon in the flowchart editor, it is automatically positioned at the point that is the next, in order to correspond to the logical program sequence. This increases the user's efficiency, since one does not have to position the icons that one has generated.

[0042] In another advantageous embodiment of the present invention, the icons of the flowchart are automatically linked together. This also increases in the operating efficiency of the user because the icons need not be linked together manually.

[0043] In another advantageous embodiment of the present invention, the flowchart is shown in a reduced or enlarged form in the display. Due to this zoom function, the diagrams are easier to comprehend for the user, and furthermore, when the user is interested in certain program sequences, he can emphasize them graphically by enlarging them.

[0044] In another advantageous embodiment of the present invention, re-translation back into flowchart representation is possible because of markings in the textual language. Due to the use of syntactical and geometric information in the form of markings, it is possible to perform such reverse translation from the textual language into the flowchart view. The reverse translation option has the advantage for the user that changes entered at the level of the textual language can be implemented directly through the flowchart editor and then become visible for the user in the flowchart diagrams. The user can then further process such retranslated programs with the help of the flowchart editor on the graphical level, and can subsequently generate control code from the said reconverted programs in the remaining procedure.

[0045] Another advantage is that a user can program both motion control functions and process control functions (programmable controller functions) in an appropriate form in a uniform programming environment. In addition, it is advantageous that the programming environment is project-sensitive, i.e., that additional dedicated language elements are made available to the user, depending on the underlying hardware or machine design.

[0046] Another advantage is that the user may use sequential as well as cyclic programming of control sequences. Since the user has access to interleaved module formation of function blocks he or she may simplify and improve the structure of his programs, because the design criteria, locality and high cohesion are easily implemented.

[0047] Another advantage is that a subprogram can automatically generate icons, containing the function interface of the such subprograms. If an original equipment manufacturer has already generated subprograms in the textual language, these subprograms can automatically expand the library of icons of the flowchart editor through appropriate icons.

[0048] An embodiment of this invention is illustrated in the drawings and explained in greater detail below.

Fig. 1 shows an engineering system, the respective run time system and the technical process to be controlled in a schematic diagram,

Fig. 2 shows elements of the engineering system and the controller and their interrelationships in a survey diagram,

Fig. 3 shows the technical program relationship between elements of the engineering system and the run time system, also in the form of a survey diagram,

Fig. 4 shows a simple diagram in flowchart representation,

Fig. 5 shows a complex diagram in flowchart representation with the control structures WHILE and IF,

Fig. 6 also shows a complex diagram in flowchart representation with the parallel branching (sync) language construct,

Fig. 7 shows a mask for setting parameters for the command "position axis,"

Fig. 8 shows a selection of language elements (icons) of the flowchart editor.

[0049] In Fig. 1, a block diagram shows that a technical process TP is controlled by the run time system RTS of an industrial controller. The connection between the run time system RTS and the controller, and the technical process TP, is bi-directional over the input/output EA. Programming of the controller and, thus, the specification of the behavior of the run time system RTS takes place in the engineering system ES. The engineering system ES contains tools for configuring, designing and programming machines, and for the control of technical processes. The programs generated in the engineering system are sent to the run time system RTS of the controller over information path I1. With regard to its hardware equipment, an engineering system ES usually comprises of a computer system with a graphical display screen (e.g., a video display unit), input means (e.g., a keyboard and mouse), a processor, working memory and secondary memory, a device for accommodating computer readable media (e.g., diskettes, CDs) and connection units for data exchange with other systems (e.g., other computer systems, controllers for technical processes) or media (e.g., the Internet). A controller usually comprises of input and output units as well as a processor and program memory.

[0050] In Fig. 2, elements of the engineering system and the controller and their interaction are illustrated in the form of a survey diagram, where the individual elements are represented in the form of rectangles, and the data storage contained in the engineering system is represented in the form of a

cylinder. Arrows (unidirectional or bi-directional) indicate the logical relationships among the elements in terms of data and sequence. The top half of Fig. 2 shows the elements of the engineering system, namely, the motion control chart (MCC) editor, the structured text (ST) compiler with programming environment, the configuration server KS and the machine design as well as a data storage. The fact that these elements belong to the engineering system is indicated by the border around them. The controller contains the code converter and program processing. The elements of the controller, which are in the lower section in Fig. 2, are outlined. Both the engineering system and the controller may also contain other elements, but, for simplicity, these are not shown.

[0051] The graphical program sequences are generated in the MCCeditor. The language elements of the editor, i.e., the icons, can be generated and represented by means of a command bar on the display screen, which is operated with the help of a mouse or other possible input means. With the help of the MCC editor, a user can link function blocks (icons) and control structures to form a flowchart, that is, the MCC editor can be used as a graphical programming tool for generating programs for motion controls and/or process controls. A text program and a textual language (usually structured text according to IEC 6-1131) are generated from the flowchart. This structured text code (ST code) is converted by the structured text converter (ST compiler), which is part of the programming environment in a processor-independent pseudo-code. This pseudo-code is loaded onto the controller where it is converted to executable processor code by the code converter. This process code is executed by the program processor within the controller. The unidirectional arrows in the left section of Fig. 2 represent the steps in code conversion or program conversion. In parallel with the three unidirectional arrows running from top to bottom, representing this conversion, three bi-directional arrows representing debugging interfaces and the possibility of program observation, run between the following elements: the MCC editor, the

ST compiler, the code converter and the program processing. Between the program processing and code converter is a debugging interface on the processor code level, i.e., on the object code level, and another debugging interface is placed between the code converter and the ST compiler. This debugging interface is on the pseudo-code level. Between the ST compiler and the MCC editor there is another debugging interface or program observation interface at the structured text level (ST code).

[0052] As additional elements of the engineering system, Figure 2 shows the machine design and a configuration server KS. In the machine design, the design of the hardware or the underlying machine is completed with the help of suitable tools, in other words, e.g. the types of axes present and the quantity specified in the machine design. This information is fed into the MCC editor through the configuration server KS. The transfer of this information is represented by the unidirectional arrows I2 and I3. In addition, the configuration server KS also contains other relevant configuration information for the system, which can also be used, for example, for licensing the respective software components.

[0053] The data storage DA, represented by a cylinder, contains three things: first, the object model generated by the MCC editor for a flowchart; second, the respective structured text; and third, the content of the data storage DA, which is the pseudo-code generated from the structured text. The data storage DA is in bi-directional connection with the MCC editor and the ST compiler, represented by the bi-directional information arrows I4 and I5.

[0054] Fig. 3 shows the existing abstraction levels from the standpoint of the program code as a survey diagram. The different program code levels are illustrated as rectangles. The top level is the MCC level, where the flowchart programs are generated. The next lower code level is the structured text level ST. One reaches the ST level from the MCC level by a corresponding code

generation as represented by an arrow from the MCC block to the ST block. Beneath the structured text level ST is the pseudo-code level. A processor-independent pseudo-code is converted by a converter from the structured text program, as represented by the arrow from the ST block to the block bearing the name pseudo-code. Beneath the pseudo-code level is the lowest code level, namely, the object code level which contains the processor code that can be executed. The object code is generated from the pseudo-code by a converter, also represented by an arrow from the pseudo-code block to the object code block. Arrows bent at a right angle lead away from the object code level back to the structured text code level ST and to the flowchart level MCC. This indicates that test activities and program tracking activities can take place on these levels on the basis of the object code. The bold double arrow between the MCC level and the ST level indicates that calls, task control commands and variable exchange functions can be sent between these two levels. The dotted line in Fig. 3 shows the borderline between the engineering system ES and the run time system RTS of the controller (S; Fig. 2). This borderline runs through the pseudo-code level, and everything above the dotted line belongs to the engineering system ES, while everything below the dotted line belongs to the run time system RTS.

[0055] In addition, Fig. 3 shows how a programmer or a user (represented by a stylized stick figure at the left edge of the figure) can introduce entries into the engineering system ES. The user can generate flowcharts on the MCC level with the help of graphical programming, or generate programs on the structured text level ST with textual programming. Both input options are represented by arrows leading from the stick figure to the MCC block or to the ST block.

[0056] The diagram according to Fig. 4 shows a simple program sequence for programming axial motions. Each flowchart begins with a start node and ends with an end node. These program-limiting symbols bear the designations "start"

and "end," respectively. Start symbols and end symbols are each represented by a rectangle with the end faces designed as semicircles. designed as two semicircles. The program commands are represented by rectangles containing a written command and a graphical symbol representing the stored command.

[0057] The flowchart symbols are usually generated in the flowchart editor by using an input bar with the help of a mouse in the flowchart editor, but other input means such as a touch pad are also conceivable. Alternatively, the system might be operated by means of a keyboard, with or without a mouse.

[0058] As the default, the flowchart symbols are directed at one another by the flowchart editor and are linked together by a line.

[0059] A synchronous axis is enabled after the start, and then the system waits for a synchronization signal, and as the next and final command of the flowchart, a cam plate, is turned on for the synchronous axis. The command sequence Fig. 4 is terminated by the end symbol.

[0060] The diagram of Fig. 5 shows a complex flowchart with control structures for a WHILE loop and for the IF statement. The WHILE and the IF statements are each represented by hexagonal honeycomb-shaped symbols. Otherwise, the same types of symbols as those already known from Figure. 4 are used in the program run illustrated in Figure 5. The flowchart begins with the start symbol and ends with the end symbol. Immediately after the start node, there is a command that starts the task "motion_3." This command is of the "start task" type. Therefore, the rectangle for this command also contains the respective corresponding symbol representing the starting of a task. The hexagonal honeycomb- shaped WHILE statement follows next in the program sequence. As long as the condition indicated in the WHILE statement is true, the commands following the WHILE statement are executed cyclically in succession. The end of the command sequence of a WHILE loop is represented by a angle

arrow leading down from the last symbol of the WHILE statement (this is the command of the type "gear synchronization off" based on a synchronous axis) and leading back to the WHILE statement on the left side of the figure. If the condition in the WHILE statement is no longer met, then the command sequence belonging to the statement is no longer executed. This is illustrated in by a rectangular connecting line leaving the WHILE symbol on the right side and bypassing the sequence of symbol commands belonging to the WHILE symbol on the right side and opening into the symbol directly following this command sequence, which is the end symbol.

[0061] However, if the WHILE condition is met, the following command sequence is processed: immediately after the WHILE statement follows a command which represents waiting for a condition. This command also contains a corresponding mnemonic graphical signal that graphically represents the waiting process graphically. This is followed by a command which starts the "motion_2" task. This command is also of the "start task" type and contains the corresponding graphical symbol. This command is followed by the IF statement, which is illustrated similarly to the WHILE statement by a hexagonal honeycomb-shaped symbol. If the IF condition is met (represented by "error <> zero"), then the command sequence is further processed in the true branch. Otherwise, if the condition is not met, the command sequence in the false branch is processed further. The next command in the true branch of the IF condition is a command that stops the "motion_2" task. This command is of the "stop task" type. It is followed by a command that stops the "motion_3" task. This command is also of the "stop task" type. These commands are also represented by respective corresponding symbols. Next in the command sequence are two "stop axis" commands. In the first such command, a rotational speed axis is stopped, and in the following command a positioning axis is stopped. These "stop axis" commands are also represented by corresponding graphical symbols. The next and also the last command relates to an axis with the name

"synchronous axis," namely, the disconnection of the gear synchronization ("gear synchronization off'). The symbols of the flowchart are connected by lines, which thus, representing the program sequence. An arrow bent at a right angle leads away from this command, representing the last command in the WHILE statement, and goes back to this WHILE statement. This represents the cyclic processing of the command sequence. In the WHILE statement, a check is performed to determine whether the condition is met. If it has been met or continues to be met, the command sequence is executed again. If it is not met, it leaves the WHILE statement and continues with the end symbol, i.e., the program run represented by the flowchart is ended.

[0062] Figure 6 shows a complex diagram in flowchart representation with the parallel branching language construction (sync). The start symbol is followed by a command that relates to a rotational speed axis, namely, "switch axis release." For this command, a graphical symbol representing this command is also shown in the command rectangle. This is again followed by a command of the "switch axis release", but this time it relates to a positioning axis; here again, the respective corresponding symbol is given. The following command is a synchronization command "wait for signal," designated as "auto" provided with the corresponding symbol. The symbol for the parallel branch (sync) follows as the next symbol. This symbol, like the WHILE and the IF statements, is also represented by a hexagonal honeycomb-shaped graphical element. All the commands arranged in the sector directly beneath the symbol for the parallel branch are started in the same interpolator cycle. This includes the "position axis" command, based on a positioning axis (this type of command also includes the respective corresponding graphical symbol) and a command of the "set output" type. This "set output" type of command is also illustrated by a rectangle, this rectangle containing the address of the output (%QB40) and the corresponding symbol for this set command (S stands for set). The commands belonging to a parallel branch symbol, that is, the commands that start within the

same interpolator cycle, are connected with a line upward to the parallel branch symbol, and at the bottom they are connected by a double line.

[0063] This horizontal double line indicates that parallel processing has been stopped and a program will wait to process the following command until all the actions in the parallel branch are concluded. Thus, this is also the end symbol of the parallel branch construction. This is followed next by a command of the "rotational speed point" type, which relates to a rotational speed axis. This is followed by two commands of the "position axis" type, each abased on positioning axis. This is again followed by a command of the "stop axis" type relating to a rotational speed axis. The rectangle representing these commands, also contains the corresponding respective graphical symbols. After a command of the "stop axis- type which relates to the aforementioned rotational speed axis, follows the end symbol.

[0064] The type of flowchart programming shown here supports different types of programming. First, a more or less true parallelism is achieved through the parallel branch symbol with the start of the respective commands in an interpolator cycle, that is programming of parallel threads is supported and their respective processing is enabled. Secondly, cyclic programming, i.e., and cyclic program processing is supported. This means that it is possible to show that only successive commands are namely, on initiation of a command, to wait until the next command until the next command is initiated and processed. The flowchart programming presented here is, thus, flexible in the way it can be applied a user and used for different applications.

[0065] Fig. 7 shows a mask for setting parameters for the "position axis" flowchart command. The designation of the corresponding command, namely, "position axis" in this case, is located in the upper left of the upper bar of the mask. The upper bar also contains two switches on its right side. The switch with a question mark provides online help, and the second switch (which is

labeled with an x) is used for closing the mask. The mask with which parameters are set (the parameterization mask) includes different input sectors. In the top input sector, the corresponding axis can be selected. With the help of an input menu (represented by an input button with a small, upside-down triangle), the corresponding axes can be selected in the input window. At the upper left of this top sector is the graphical symbol belonging to this command, an upside-down triangle with a dark horizontal line at the center, and other small lines angled downward at each end of this line. The next and largest sector of the parameterization mask represents the possibility of parameter input. The parameters differ according to the command. They are sorted logically by means of task bar options which are arranged on a task bar , as is customary in the conventional program interfaces. The first page (in Fig. 7 this page can be shown by selecting the task bar option "parameter") usually contains the parameters which absolutely must be indicated for setting of parameters of the command. An unconditional parameter for the "position axis" command would be, for example, the target position of an axial motion.

[0066] The number and significance of the task bar options also varies according to the command. It can be seen in Fig. 7 that a "dynamic" task bar option is also present for the "position axis" command in addition to the "parameter" task bar option. With this task bar option, entries regarding the rate of change and acceleration as well as the velocity profile can be made for the description of the dynamic behavior. These inputs can be made through input fields and the respective menus. In this case, the trapezoidal shape was selected as the velocity profile. This shape has also been represented graphically in a stylized manner at the center of this input sector. In the lower input sector of the parameterization mask which follows this, additional inputs, e.g., for the transitional behavior, can be made. In this case, "detaching" was entered for the transitional behavior. In addition, waiting conditions can be entered by putting a check in the "wait" box. Additional entries for this

synchronization can be made in a respective input window. In the example in Fig. 7, "position window reached" has been entered for this item. The entries are also supported by axial profiles which are represented in a stylized manner. The lower end of the parameterization axis consists of four input buttons, namely, an "okay" button, a "terminate" button, an "accept" button and a "help" button. By using these input buttons, users may either accept the entries, confirm them, discard them or call up input help. With the help of the waiting conditions, so-called step enabling conditions can be specified by a user to synchronize the functions (e.g.., reference point approach or axial positioning) or their interaction.

[0067] There are particular parameterization masks for commands that can be entered and processed with the help of the flowchart editor. Thus, the user is supported in programming motion and control sequences with the help of these parameterization masks in a context sensitive manner.

[0068] The diagram according to Fig. 8 shows a selection of language elements (icons) of the flowchart editor. These language elements represent commands that can be used for graphical programming using the flowchart editor. The motion control flowchart editor supports the following classes of commands and makes available appropriate symbols available for for the individual commands of these following classes: start commands, stop commands, positioning commands, synchronism commands and probe commands, and software commands, wait commands, task control commands, commands for manipulation of variables, and other general commands. In addition, motion control flowchart editor makes available additional graphical control structures available for graphical program execution.